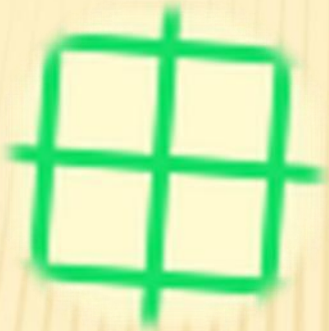
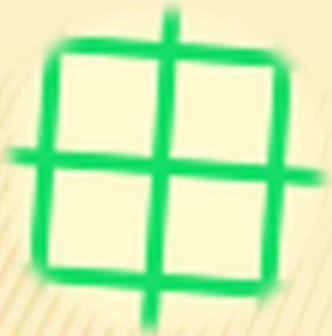
Program Grapher



Terry Nixon, Haris Ribic, K. Drew Saxton, Gabe Shreckengost

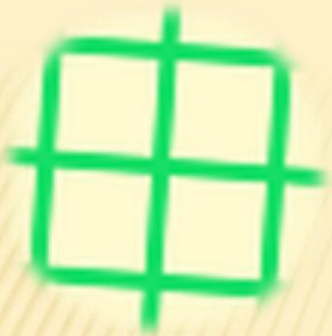
Presented December 9, 2009

Gannon University



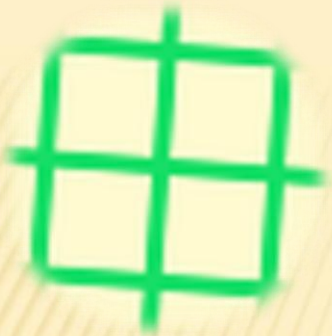
Requirements

- Project Planning
  - Carried out using the Volere Requirements Specification Template
- Purpose of the Project:
  - To develop software capable of automatically generating program graphs from Java source code



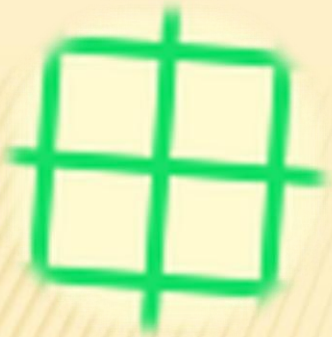
Specifications

- Limitations of the Program Grapher:
  - Can only diagram Java code
  - Java code must be formatted according to Sun Microsystems' Java Code Conventions
  - Can only handle sequential code (no objects or methods)



Parser

- Reads in line by line from Java source file
- Indentifies
  - Structures
    - if, else, else if, do, for, while, switch(case, break, default)
  - Types
    - int, float, boolean, char, string, long, byte
  - Variable definitions i.e. y = 3 or int y = 6
  - … but ignores
    - White spaces, non-executing statements (package, import)
    - Variable declarations i.e. int x
    - Comments “//” or “/*” + “*/”



Parser

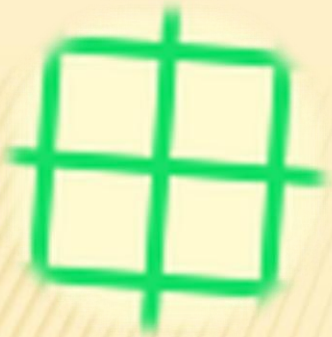
- Creates a dynamic list of strings
  - Information about each line
    - first it looks for “main” then for
      - “while” and “endWhile”
      - “do” and “endDo”
      - “if” and “endIf”
      - “ignore”
- Nested structures dynamically tracked in a stack
  - LIFO
- List is passed on to make Adjacency Matrix

Parser

```
while
{
        if (x < y)
        {
                do_this()
        }
        else
        {
                do_that()
        }
}
```

Program

```
// Project 1: Triangle Program Testing using Manual Driver and JUnit
package triangle;
public class TriangleDriver {

    public static void main (String[] args) {

        int ctr = 0;
        while(ctr < 10)
        {
            ctr++;
            ctr++;
            ctr++;

            if(ctr < 10)
            {
                ctr--;
            }
        }
        // jjj
        if(ctr < 10)
        {
            ctr--;
        }
            else
        {
            // kkk
            ctr--;
        }

        switch (item)
        {
            case "f": doThis; break;
            case "f": doThat; break;
            default : doNothing; break;
        }

        do
        {
            ctr++;
            ctr++;
        } while (ctr < 50);
        ctr = 0;
    }
}
```

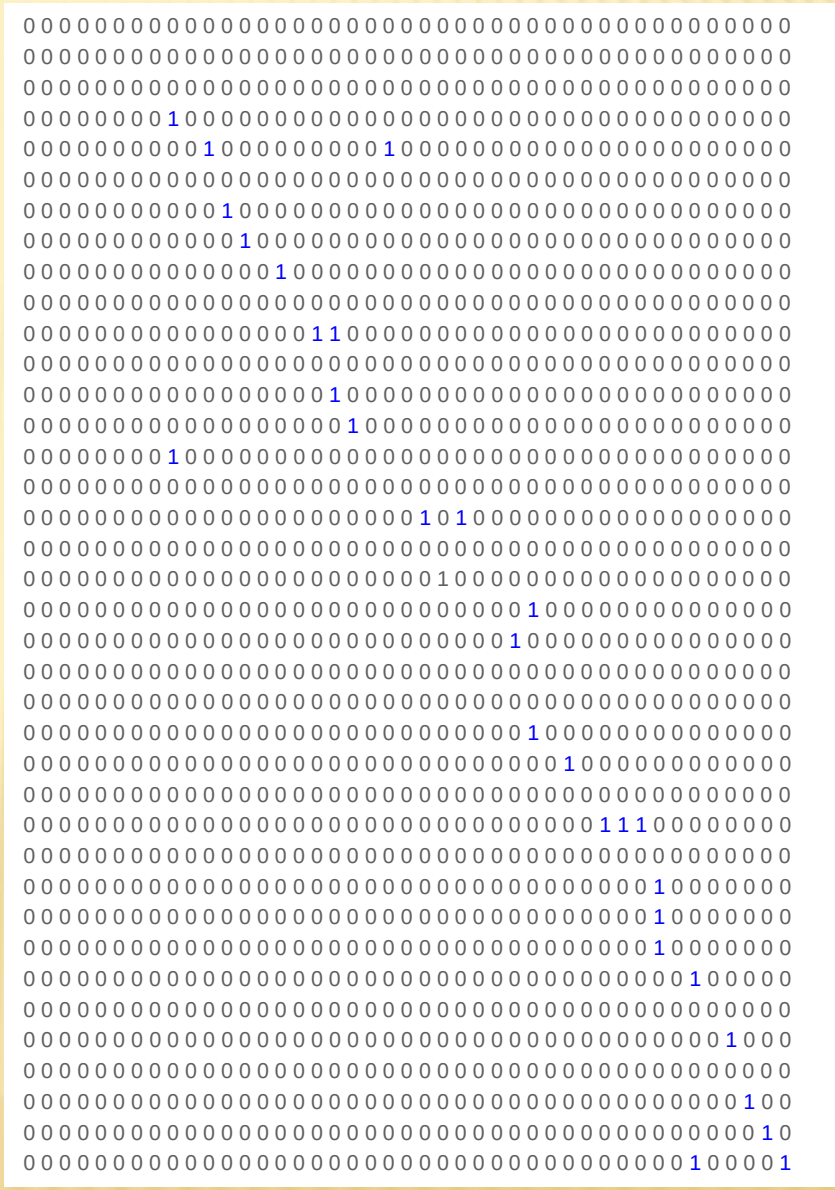


Parser Output

Matrix Generator Input

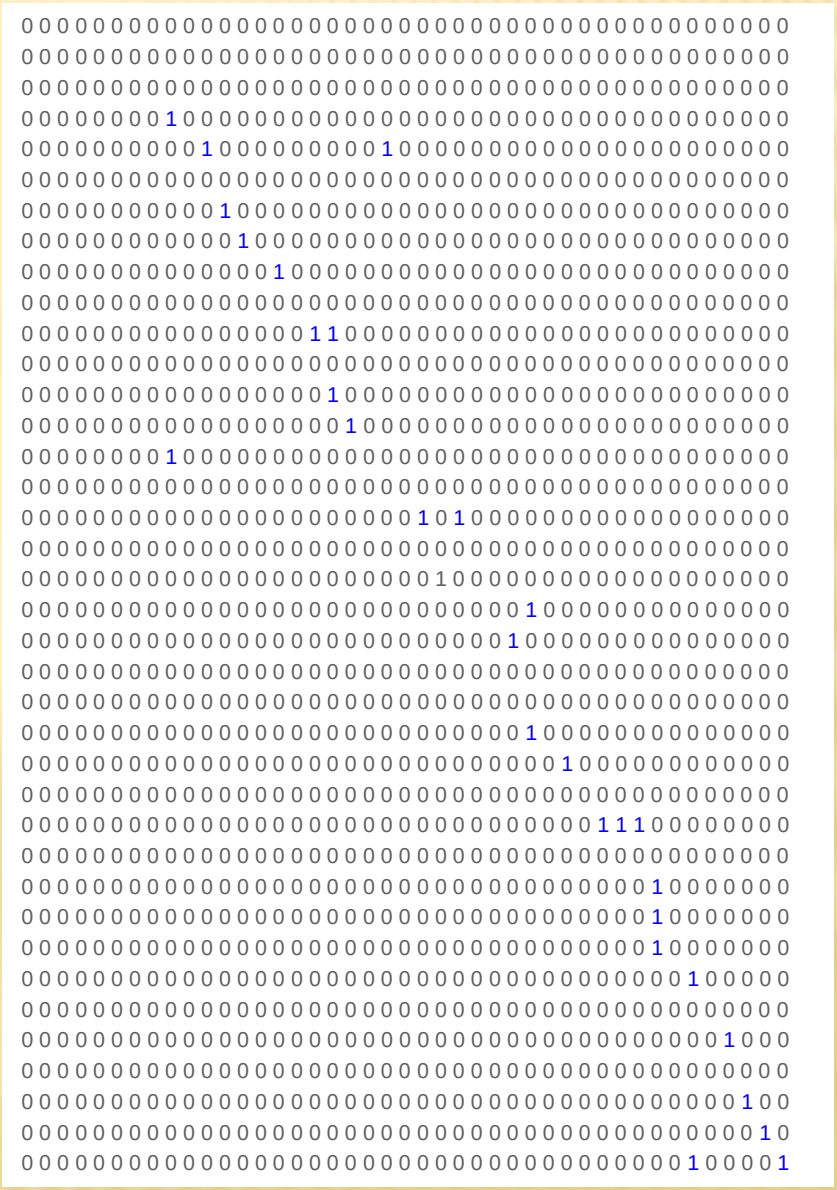
ignore
ignore
ignore
ignore
Main
ignore
sequence
Loop
ignore
sequence
sequence
sequence
ignore
If
ignore
sequence
EndIf
EndLoop
ignore
If
ignore
sequence
EndIf
Else
ignore
ignore
sequence
EndElse
ignore
Switch
ignore
Case
Case
Default
EndSwitch
ignore
DoLoop
ignore
sequence
sequence
EndDoLoop
sequence
end
end

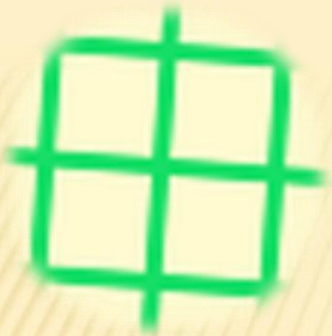
Matrix Generator Output (Adjacency Matrix)



```
ignore
ignore
ignore
ignore
Main
ignore
sequence
Loop
ignore
sequence
sequence
sequence
ignore
If
ignore
sequence
EndIf
EndLoop
ignore
If
ignore
sequence
EndIf
Else
ignore
ignore
sequence
EndElse
ignore
Switch
ignore
Case
Case
Default
EndSwitch
ignore
DoLoop
ignore
sequence
sequence
EndDoLoop
sequence
end
end
```

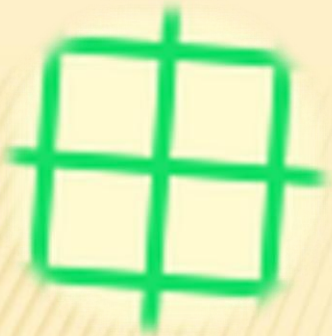
How?





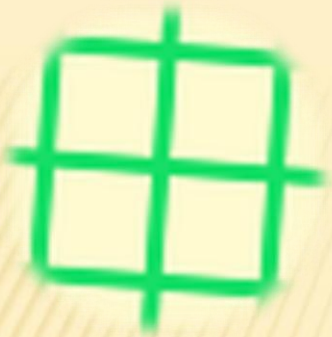
Parser Keywords

ignore	
Main	
sequence	- Any executable
Loop	- Loop Pre-Test (For & While)
EndLoop	
DoLoop	- Loop Post-Test (Do)
EndDoLoop	
If	
EndIf	
Else	
EndElse	
Else If	
EndElse If	
Switch	
Case	
Default	
EndSwitch	
end	- End of program



Matrix Generator Variables

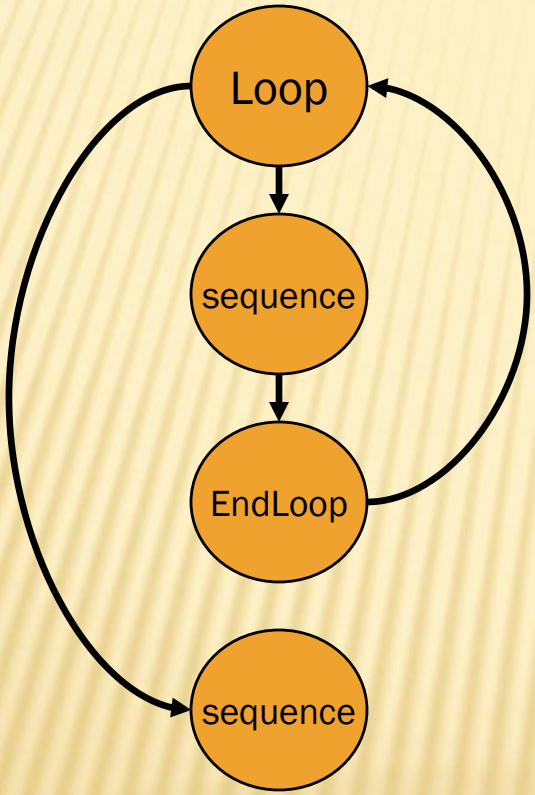
```
int index = 1;                                  //Program Code line number
int previous = -1;                              //Previous "executable" line number
int numOfIfs = 0;                               //Number of If statements
int numOfEndIfs = 0;                            //Number of end if statements
int numOfNestedLoops = 0;                        //Nested Loop counter
int numOfCaseStatements = 0;                    //The Switch case statement counter
int switchLineNumber = -1;                      //Line number of the switch statement

int[] ifLineNumber = new int[20];                //Line number of if statement - Maximum of 20
int[] endIfLineNumber = new int[20];             //Line number of endif statement
int[] loopLineNumber = new int[20];              //Line number of loop statement
int[] caseEndLineNumber = new int[20];  //Last line number of each case - Maximum of 20

bool[] isElse = new bool[20];                   //Flag to indicate if there is a matching else
bool[] endIfFlag = new bool[20];                //Flag to indicate if there is an endif
bool exitLoop = false;                          //Flag to exit the loop - Reached the program end

string keyword;                                 //Hold the keyword from the parser
```

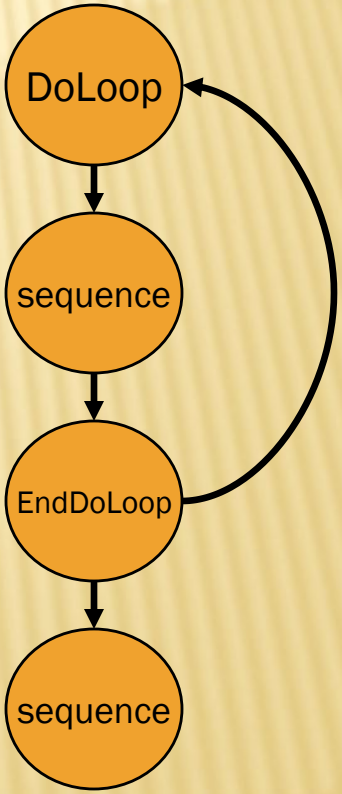


Matrix Generator Structure

```
do
{
    keyword = inputFile[index - 1];    //Get the keyword from the parser output
    switch (keyword)
    {
            case "ignore":
            case "Main":
                    index++;            //Increase index & do nothing with the adjacency matrix
                    break;
                .
                .
                .
            default:
                    Console.WriteLine("ERROR - Incorrect type of statement");
                    exitLoop = true;            //Exit the program because of error
                    throw new Exception();
     }

} while (!exitLoop);   //Loop until error or reach the end of the program

if (adjacencyMatrix.IsEmpty())
{
            throw new Exception();
}
return adjacencyMatrix;
```

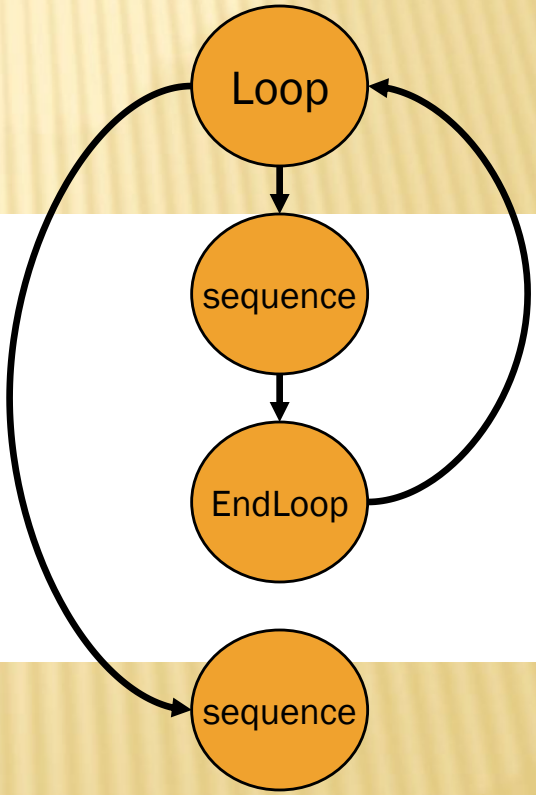
Matrix Generator Loops





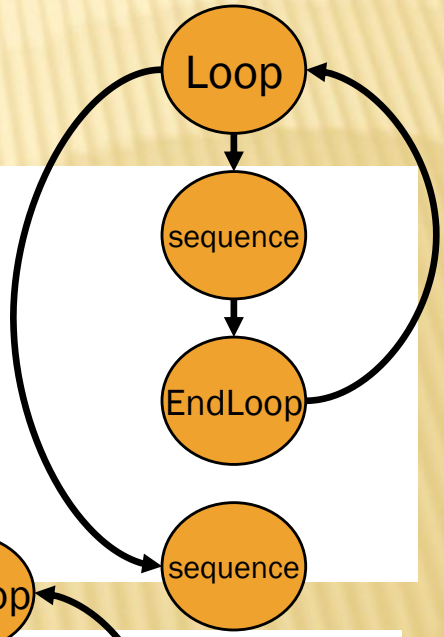
Matrix Generator Loop

```
case "Loop":
        //Was an endIf the previous statement?
        else
        {
                adjacencyMatrix.SetValid(previous, index);
        }
        loopLineNumber[numOfNestedLoops] = index;        //Save line number
        numOfNestedLoops++;         //Increase Counter
        previous = index;
        index++;
        break;
```

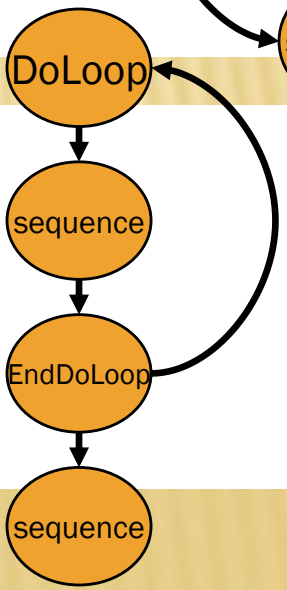


Matrix Generator
End Loops

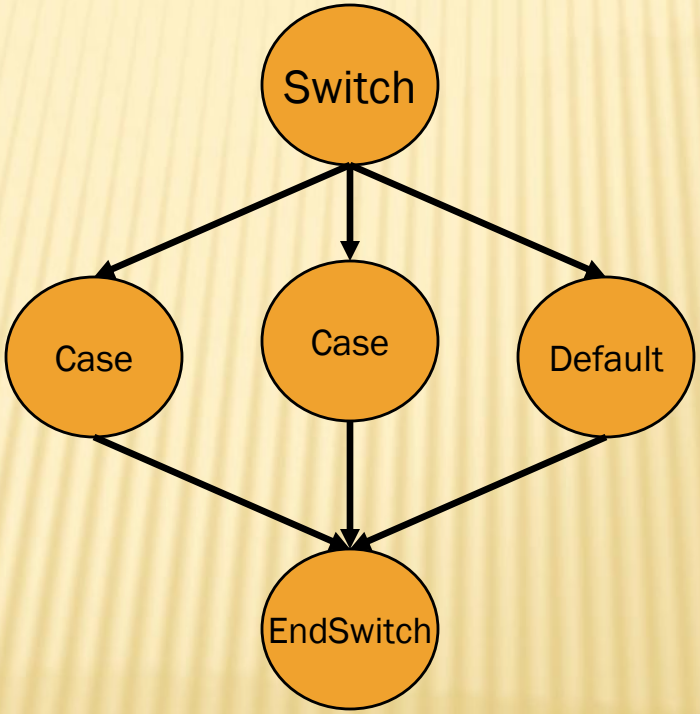
```
case "EndLoop":
        //Was an endIf the previous statement?
        adjacencyMatrix.SetValid(previous, index);
        adjacencyMatrix.SetValid(index, loopLineNumber[numOfNestedLoops - 1]);
        numOfNestedLoops--;
        previous = loopLineNumber[numOfNestedLoops - 1];
        index++;
        break;
```




```
case "EndDoLoop":
        //Was an endIf the previous statement?
        adjacencyMatrix.SetValid(previous, index);
        adjacencyMatrix.SetValid(index, loopLineNumber[numOfNestedLoops - 1]);
        numOfNestedLoops--;
        previous = index;
        index++;
        break;
```

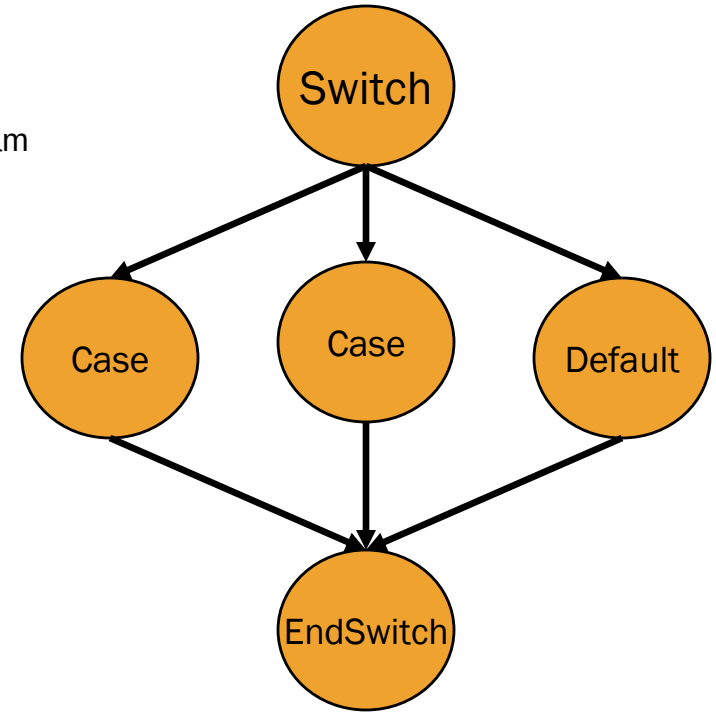


Matrix Generator Switch



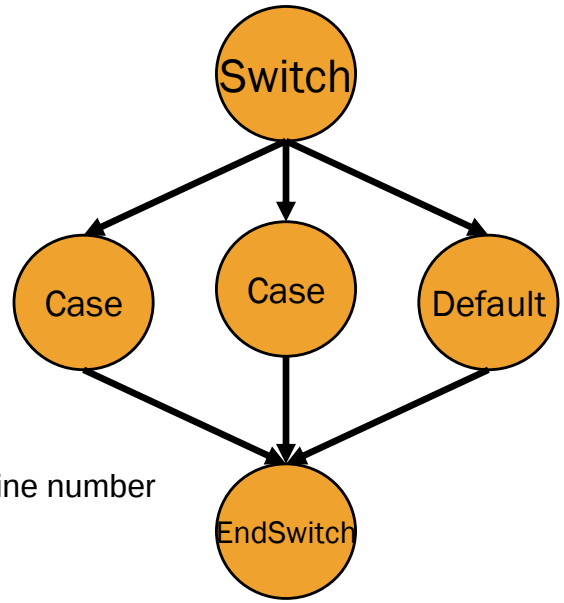
Matrix Generator Switch

```
case "Switch":
        //Was an endIf the previous statement?
        else
        {
                if (previous == -1)     //If the 1st statement in the java program
                {
                        previous = index;
                }
                else
                {
                        adjacencyMatrix.SetValid(previous, index);
                }
                switchLineNumber = index;   //Save line number
                previous = index;
        }
        index++;
        break;
```



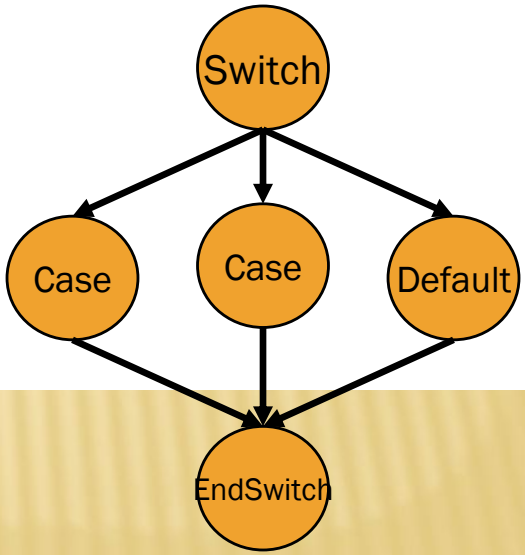
Matrix Generator Case and Default

```
case "Case":
case "Default":
      if (numOfCaseStatements == 0)   //If 1st case statement
      {
            numOfCaseStatements++;      //Increase counter
            adjacencyMatrix.SetValid(previous, index);
            previous = index;
            index++;
      }
      else
      {
            caseEndLineNumber[numOfCaseStatements - 1] = previous;    //Save line number
            numOfCaseStatements++;                                    //Increase counter
            adjacencyMatrix.SetValid(switchLineNumber, index);
            previous = index;
            index++;
      }
    break;
```



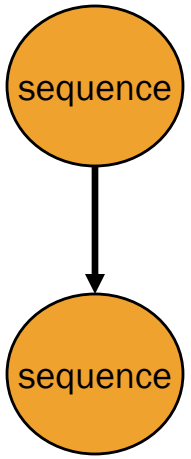
Matrix Generator EndSwitch

```
case "EndSwitch":
        //Was an endIf the previous statement?

        for (int i = 0; i < numOfCaseStatements - 1; i++)     //Loop through the number of case statements
        {
                adjacencyMatrix.SetValid(caseEndLineNumber[i], index);
        }
        adjacencyMatrix.SetValid(previous, index);
        numOfCaseStatements = 0;
        previous = index;
        index++;
        break;
```



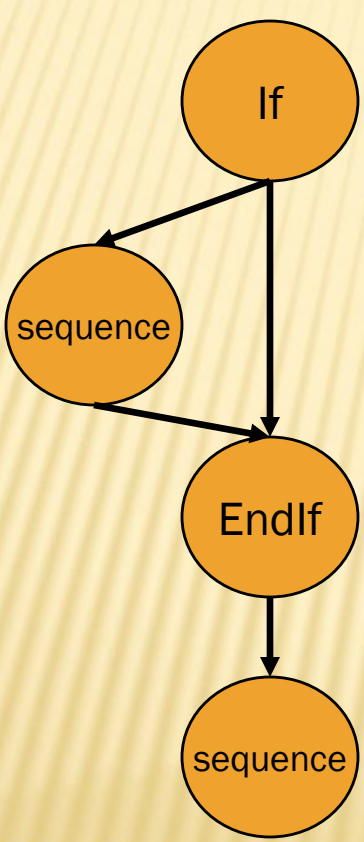
Matrix Generator sequence and end

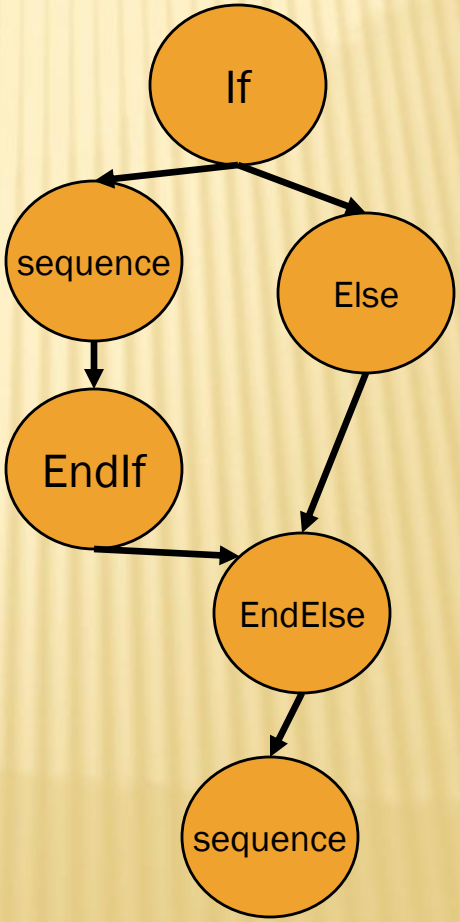
```
case "sequence":
case "end":
        //Was an endIf the previous statement?
        else
        {
            if (previous == -1)    //If this is the first in the graph
            {
                previous = index;   //Set previous
             }
             else
             {
                    adjacencyMatrix.SetValid(previous, index);
                    previous = index;
              }
          }
          if (keyword.Equals("end"))
          {
               exitLoop = true;    //Get out of loop - End of program
          }
          index++;
          break;
```



Matrix Generator

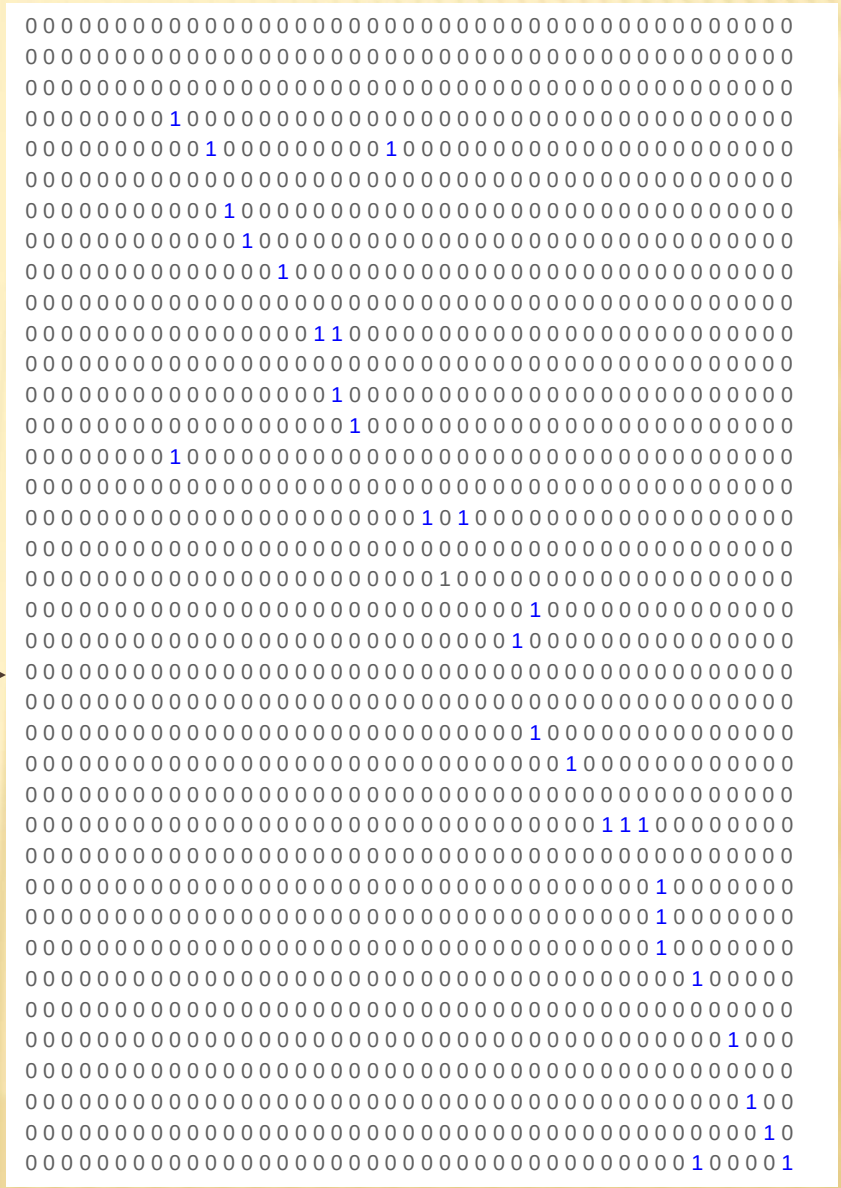
If

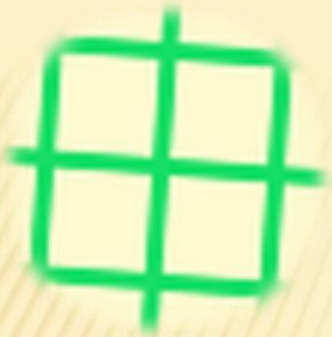




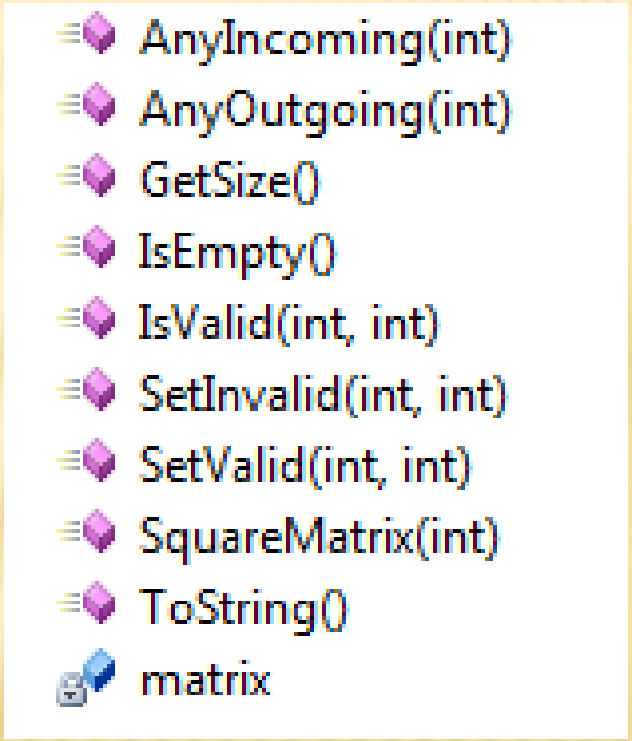
ignore
ignore
ignore
ignore
Main
ignore
sequence
Loop
ignore
sequence
sequence
sequence
ignore
If
ignore
sequence
EndIf
EndLoop
ignore
If
ignore
sequence
EndIf
Else
ignore
ignore
sequence
EndElse
ignore
Switch
ignore
Case
Case
Default
EndSwitch
ignore
DoLoop
ignore
sequence
sequence
EndDoLoop
sequence
end
end

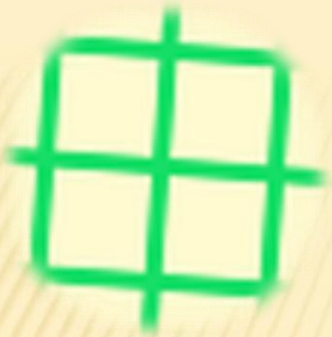
So that is how!





Square Matrix Class

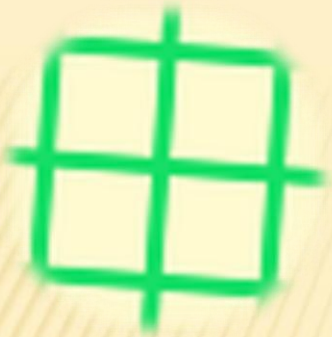




Square Matrix Class

```
private int[,] matrix;

/// <summary>
/// Creates an n-by-n square matrix.
/// </summary>
public SquareMatrix(int n)
{
    matrix = new int[n, n];
}

/// <summary>
/// Sets the value at location [i,j]
/// </summary>
public void SetValid(int i, int j)
{
    matrix[i - 1, j - 1] = 1;
}

/// <summary>
/// Sets the value at location [i,j]
/// </summary>
public void SetInvalid(int i, int j)
{
    matrix[i - 1, j - 1] = 0;
}
```

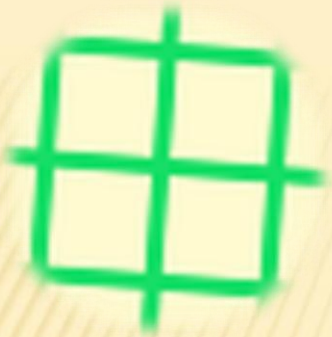
```
/// <summary>
/// Determines if the value at location [i,j]
/// </summary>
public bool IsValid(int i, int j)
{
    if (matrix[i - 1, j - 1] == 1)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/// <summary>
/// Determines if there are any incoming edges
/// </summary>
public bool AnyIncoming(int vertex)
{
    for (int i = 0; i < GetSize(); i++)
    {
        if (matrix[i, vertex - 1] == 1)
        {
            return true;
        }
    }

    return false;
}
```



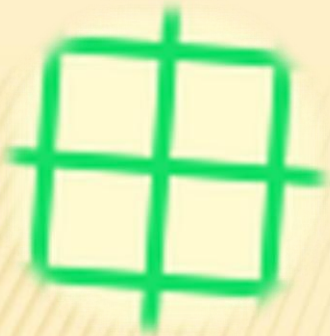
Square Matrix Class

```
/// <summary>
/// Determines if there are any outgoing edges to the given
/// </summary>
public bool AnyOutgoing(int vertex)
{
    for (int i = 0; i < GetSize(); i++)
    {
        if (matrix[vertex - 1, i] == 1)
        {
            return true;
        }
    }

    return false;
}

/// <summary>
/// Determines if the matrix is empty (contains all 0s).
/// </summary>
public bool IsEmpty()
{
    for (int i = 0; i < GetSize(); i++)
    {
        for (int j = 0; j < GetSize(); j++)
        {
            if (matrix[i, j] == 1)
            {
                return false;
            }
        }
    }

    return true;
}
```

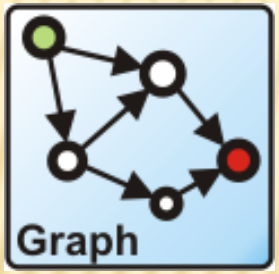
```
/// <summary>
/// Gets the dimensions of the square matrix.
/// </summary>
public int GetSize()
{
    return (int)Math.Sqrt(matrix.Length);
}

/// <summary>
/// Returns the matrix as a string object.
/// </summary>
public override String ToString()
{
    String printedMatrix = "";

    for (int i = 0; i < GetSize(); i++)
    {
        for (int j = 0; j < GetSize(); j++)
        {
            printedMatrix += matrix[i, j] + " ";
        }
        printedMatrix += "\n";
    }

    return printedMatrix;
}
```



Architecture

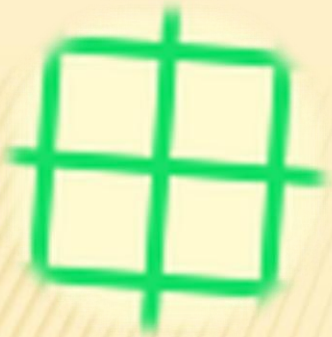


Graph#



Open source graph layout framework

http://graphsharp.codeplex.com/



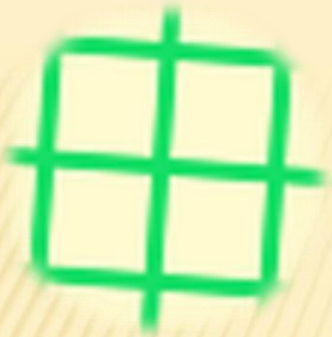
Graph#

Supported layout algorithms:

- Fruchterman-Reingold
- Kamada-Kawai
- ISOM
- LinLog
- Simple Tree
- Simple Circle
- Sugiyama
- Compound FDP

Graph#

http://graphsharp.codeplex.com/



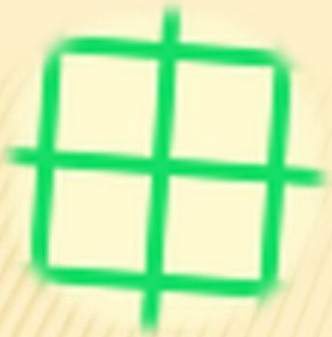
Graph#

Additional algorithms supported:

- Overlap removal algorithms
- Edge layout algorithms
- Highlight algorithms

Graph#

http://graphsharp.codeplex.com/



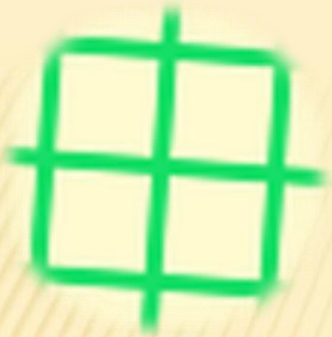
Graph#

WPF control:

- Supports all of the layout algorithms
- Vertex dragging
- Supports templates for vertices and edges
- Automatic relayout after the graph changes
- Animated position changes

Graph#

http://graphsharp.codeplex.com/

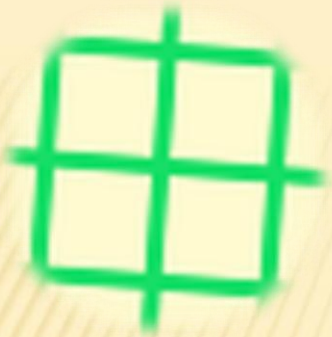


WPF

Windows Presentation Foundation

- Graphical subsystem for rendering user interfaces in Windows-based applications
- Built on DirectX, which provides hardware acceleration for transparency, gradients, etc.
- WPF provides a clear separation between the user interface and the business logic.

http://en.wikipedia.org/wiki/Windows_Presentation_Foundation



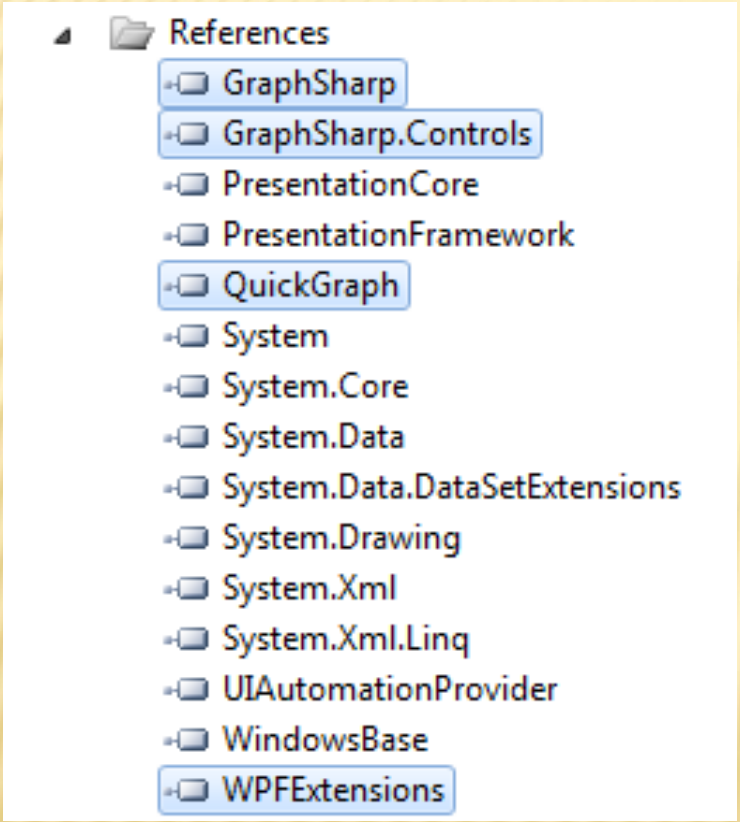
XAML

Extensible Application Markup Language

- New markup language, which used to define UI elements and relationships with other UI elements
- Extended from XML

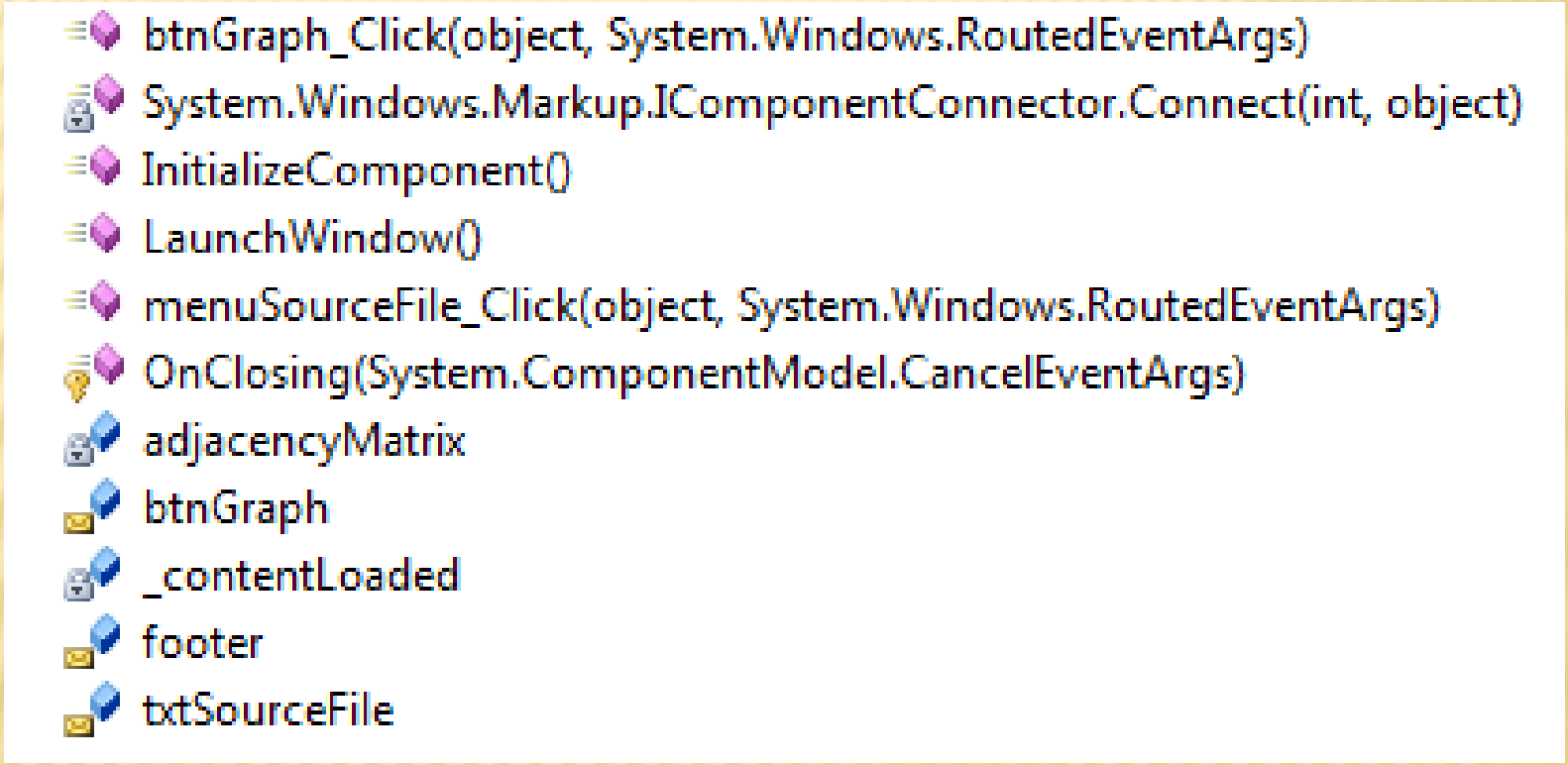
http://en.wikipedia.org/wiki/XAML

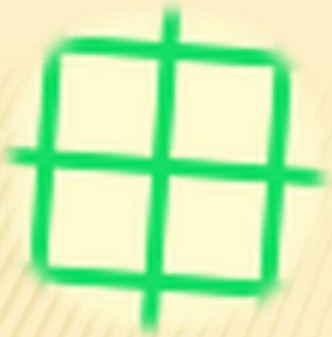
Required References



1. GraphSharp.dll
2. GraphSharp.Controls.dll
3. QuickGraph.dll
4. WPFExtensions.dll

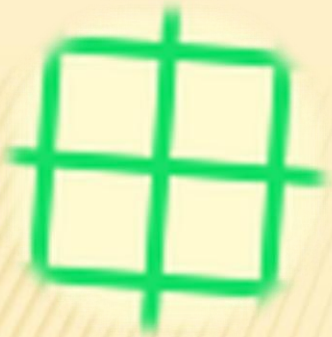
Launch Window





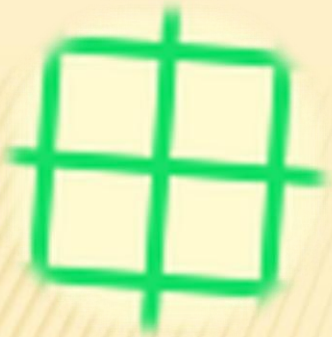
Launch Window

```
public LaunchWindow()
{
    System.Drawing.Bitmap icon = global::ProgramGrapher.Properties.Resources.graph;

    MemoryStream iconStream = new MemoryStream();
    icon.Save(iconStream, System.Drawing.Imaging.ImageFormat.Png);
    iconStream.Seek(0, SeekOrigin.Begin);
    BitmapFrame newIcon = BitmapFrame.Create(iconStream);
    this.Icon = newIcon;

    InitializeComponent();

    btnGraph.IsEnabled = false;
}

private SquareMatrix adjacencyMatrix;
```



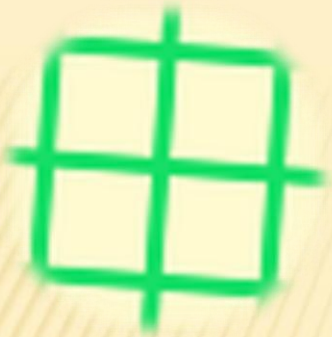
Launch Window

```
public void menuSourceFile_Click(object sender, RoutedEventArgs e)
{
    Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
    dlg.FileName = "";
    dlg.DefaultExt = ".java";
    dlg.Filter = "JAVA File (.java)|*.java";

    Nullable<bool> result = dlg.ShowDialog();

    if (result == true)
    {
        string fileName = dlg.FileName;

        string[] fileNameParts = dlg.FileName.Split('\\');

        footer.Text = "Current file: " + fileNameParts.Last<string>();

        btnGraph.IsEnabled = false;

        try
        {
            adjacencyMatrix = new MatrixGenerator(fileName).CreateAdjacencyMatrix();
        }
        catch (Exception)
        {
            txtSourceFile.Text = "Source file could not be opened!\n\nPlease check the given source file for errors.";
            return;
        }

        txtSourceFile.Text = "Source file successfully opened!\n\nYou can now view the program graph for the given source file.";

        btnGraph.IsEnabled = true;
    }
}
```



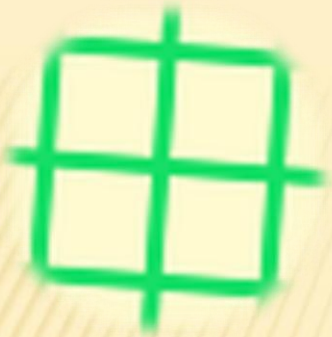
Launch Window

```
        public void btnGraph_Click(object sender, RoutedEventArgs e)
        {
            GraphWindow graphWindow1 = new GraphWindow(adjacencyMatrix);
            graphWindow1.Show();
        }

        protected override void OnClosing(System.ComponentModel.CancelEventArgs e)
        {
            base.OnClosing(e);
            if (MessageBox.Show("Are you sure that you want to close the application?", "Confirm Exit",
MessageBoxButton.OKCancel) != MessageBoxResult.OK)
            {
                e.Cancel = true;
            }
        }
```



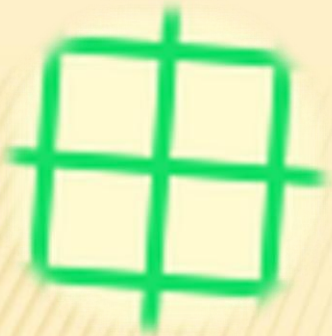
Graph Window

- System.Windows.Markup.IComponentConnector.Connect(int, object)
- CreateGraphToVisualize(ProgramGrapher.SquareMatrix)
- GraphWindow(ProgramGrapher.SquareMatrix)
- InitializeComponent()
- GraphToVisualize
- _contentLoaded
- graphLayout
- _graphToVisualize
- root
- zoomControl



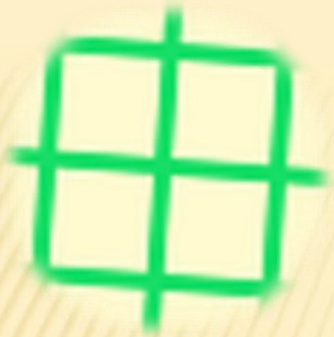
Graph Window

```
private IBidirectionalGraph<object, IEdge<object>> _graphToVisualize;

public IBidirectionalGraph<object, IEdge<object>> GraphToVisualize
{
    get { return _graphToVisualize; }
}
```



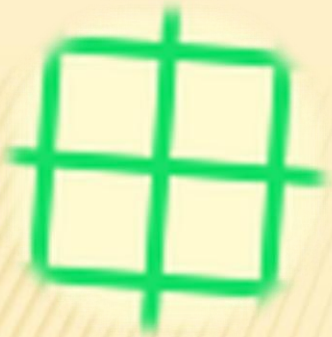
Graph Window

```
public GraphWindow(SquareMatrix adjacencyMatrix)
{
    System.Drawing.Bitmap icon = global::ProgramGrapher.Properties.Resources.graph;

    MemoryStream iconStream = new MemoryStream();
    icon.Save(iconStream, System.Drawing.Imaging.ImageFormat.Png);
    iconStream.Seek(0, SeekOrigin.Begin);
    BitmapFrame newIcon = BitmapFrame.Create(iconStream);
    this.Icon = newIcon;

    CreateGraphToVisualize(adjacencyMatrix);

    InitializeComponent();

    this.WindowState = WindowState.Maximized;
}
```



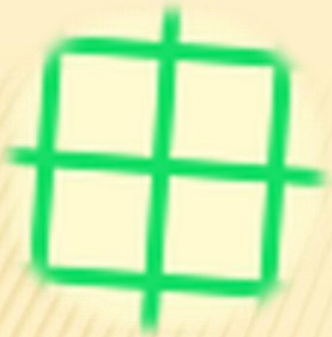
Graph Window

```
private void CreateGraphToVisualize(SquareMatrix adjacencyMatrix)
{
    var g = new BidirectionalGraph<object, IEdge<object>>();

    //add vertices to the graph
    for (int i = 0; i < adjacencyMatrix.GetSize(); i++)
    {
        if (adjacencyMatrix.AnyIncoming(i + 1) || adjacencyMatrix.AnyOutgoing(i + 1))
        {
            g.AddVertex((i + 1).ToString());
        }
    }

    //add edges to the graph
    for (int i = 0; i < adjacencyMatrix.GetSize(); i++)
    {
        for (int j = 0; j < adjacencyMatrix.GetSize(); j++)
        {
            if (adjacencyMatrix.IsValid(i + 1, j + 1))
            {
                g.AddEdge(new Edge<object>((i + 1).ToString(), (j + 1).ToString()));
            }
        }
    }

    _graphToVisualize = g;
}
```



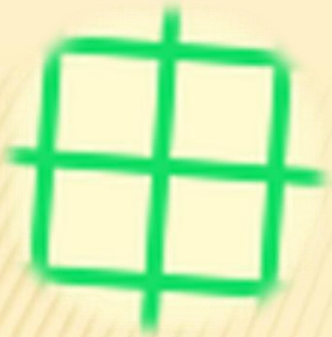
Quick Graph

- Degree(TVertex)
- InDegree(TVertex)
- InEdge(TVertex, int)
- InEdges(TVertex)
- IsInEdgesEmpty(TVertex)
- TryGetInEdges(TVertex, out System.Collections.Generic.IEnumerable<TEdge>)



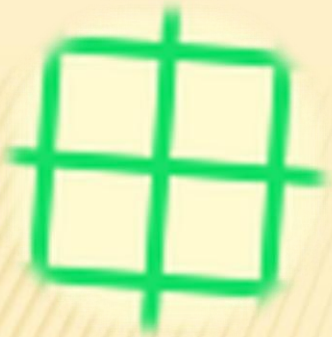
Quick Graph

```
namespace QuickGraph
{
    public interface IBidirectionalGraph<TVertex, TEdge> : IVertexAndEdgeListGraph<TVertex,
TEdge>, IVertexListGraph<TVertex, TEdge>, IIncidenceGraph<TVertex, TEdge>,
IImplicitGraph<TVertex, TEdge>, IEdgeListGraph<TVertex, TEdge>, IGraph<TVertex, TEdge>,
IEdgeSet<TVertex, TEdge>, IVertexSet<TVertex>, IImplicitVertexSet<TVertex> where TEdge :
global::QuickGraph.IEdge<TVertex>
    {
        int Degree(TVertex v);
        int InDegree(TVertex v);
        TEdge InEdge(TVertex v, int index);
        IEnumerable<TEdge> InEdges(TVertex v);
        bool IsInEdgesEmpty(TVertex v);
        bool TryGetInEdges(TVertex v, out IEnumerable<TEdge> edges);
    }
}
```



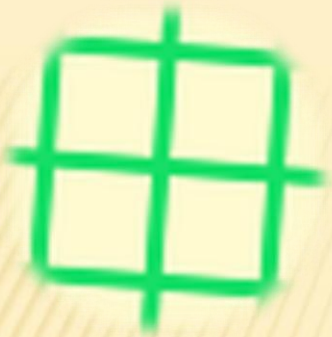
Launch Window

```
<Window x:Class="ProgramGrapher.LaunchWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Program Grapher" Height="325" Width="500" ResizeMode="NoResize">

    <DockPanel HorizontalAlignment="Stretch" Margin="0,0,0,0" Width="Auto">

        <Menu Width="Auto" Height="Auto" DockPanel.Dock="Top" Background="#7d0c00">
            <MenuItem Header="Open File" Click="menuSourceFile_Click" Foreground="White" FontSize="15" />
        </Menu>

        <StatusBar Width="Auto" Height="25" Background="#00395c" DockPanel.Dock="Bottom">
            <TextBlock x:Name="footer" Width="Auto" Height="Auto" Foreground="White" Text="Open a JAVA source file." FontSize="15" />
        </StatusBar>

        <GroupBox DockPanel.Dock="Left" Width="250" Height="200" Header="Source File" Margin="10, 5, 5, 5" FontSize="20">
            <TextBlock x:Name="txtSourceFile" Height="Auto" Width="Auto" Margin="5, 15, 5, 15" TextWrapping="Wrap" FontSize="15" />
        </GroupBox>

        <GroupBox DockPanel.Dock="Right" Width="190" Height="200" Header="Program Graph" Margin="5, 5, 5, 5" FontSize="20">
            <Button x:Name="btnGraph" Content="View Graph" Click="btnGraph_Click" Height="50" Width="Auto" Margin="5, 5, 5, 5" FontSize="15" />
        </GroupBox>

    </DockPanel>

</Window>
```



Graph Window

```
<Window x:Class="ProgramGrapher.GraphWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:graphsharp="clr-namespace:GraphSharp.Controls;assembly=GraphSharp.Controls"
    xmlns:zoom="clr-namespace:WPFExtensions.Controls;assembly=WPFExtensions"
    Title="Program Grapher" Height="300" Width="300" x:Name="root">

    <Grid>

        <zoom:ZoomControl x:Name="zoomControl">

            <graphsharp:GraphLayout x:Name="graphLayout"
                                    Graph="{Binding ElementName=root,Path=GraphToVisualize}"
                                    LayoutAlgorithmType="EfficientSugiyama"
                                    HighlightAlgorithmType="Simple" />

        </zoom:ZoomControl>

    </Grid>

</Window>
```



Sugiyama Algorithm

Designed by Sugiyama, Tagawa, and Toda in 1981

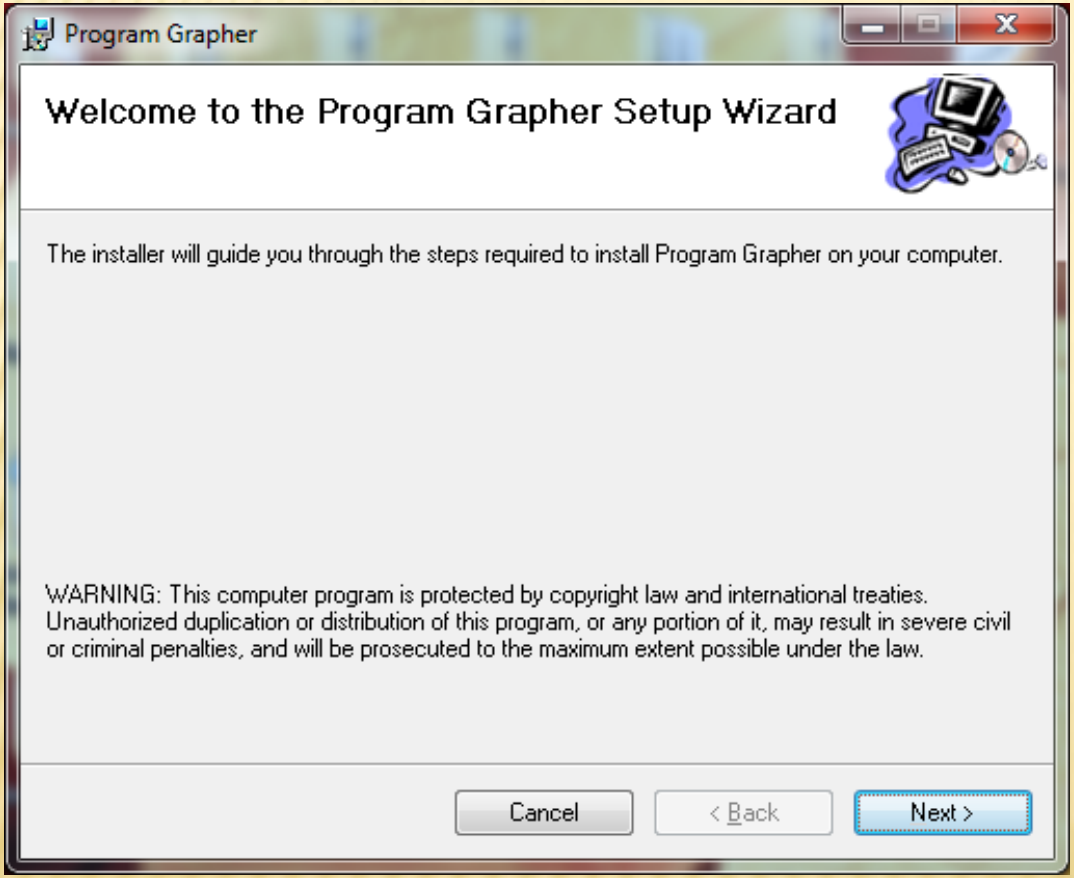
Input: Hierarchical directed graph

Draws graph in four steps:

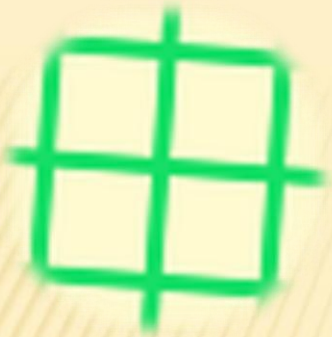
1. Graph is transformed into a “proper” hierarchy (if necessary)
2. Vertices at each level are ordered to reduce the number of edge crossings
3. The horizontal position of each vertex is manipulated to reduce the length of the edges
4. Graph is drawn

http://www.ics.uci.edu/~ses/papers/grafdraw.pdf

Installation


Program Grapher
Welcome to the Program Grapher Setup Wizard
The installer will guide you through the steps required to install Program Grapher on your computer.
WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.
Cancel
< Back
Next >

1. Installs all requirements
2. Installs application
3. Creates desktop and start menu shortcuts
4. Includes user manual



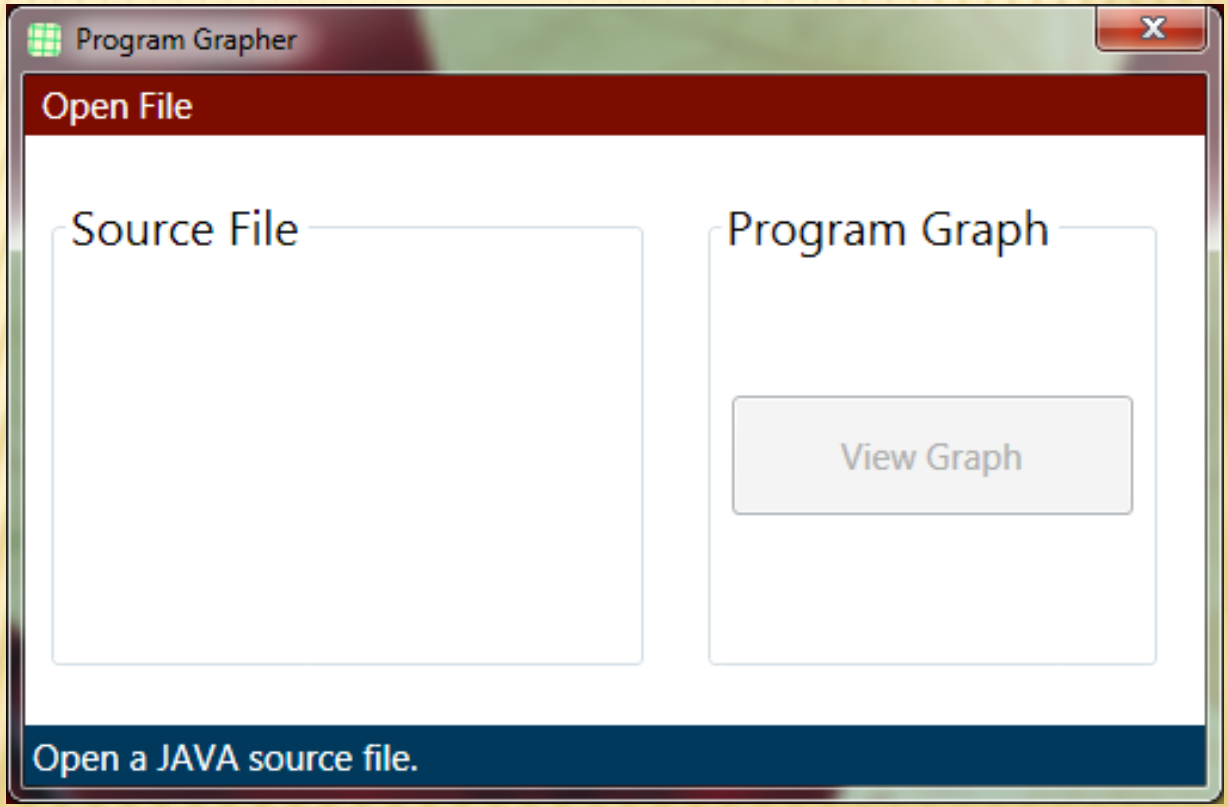
Deployment

URL: http://drop.io/programgrapher

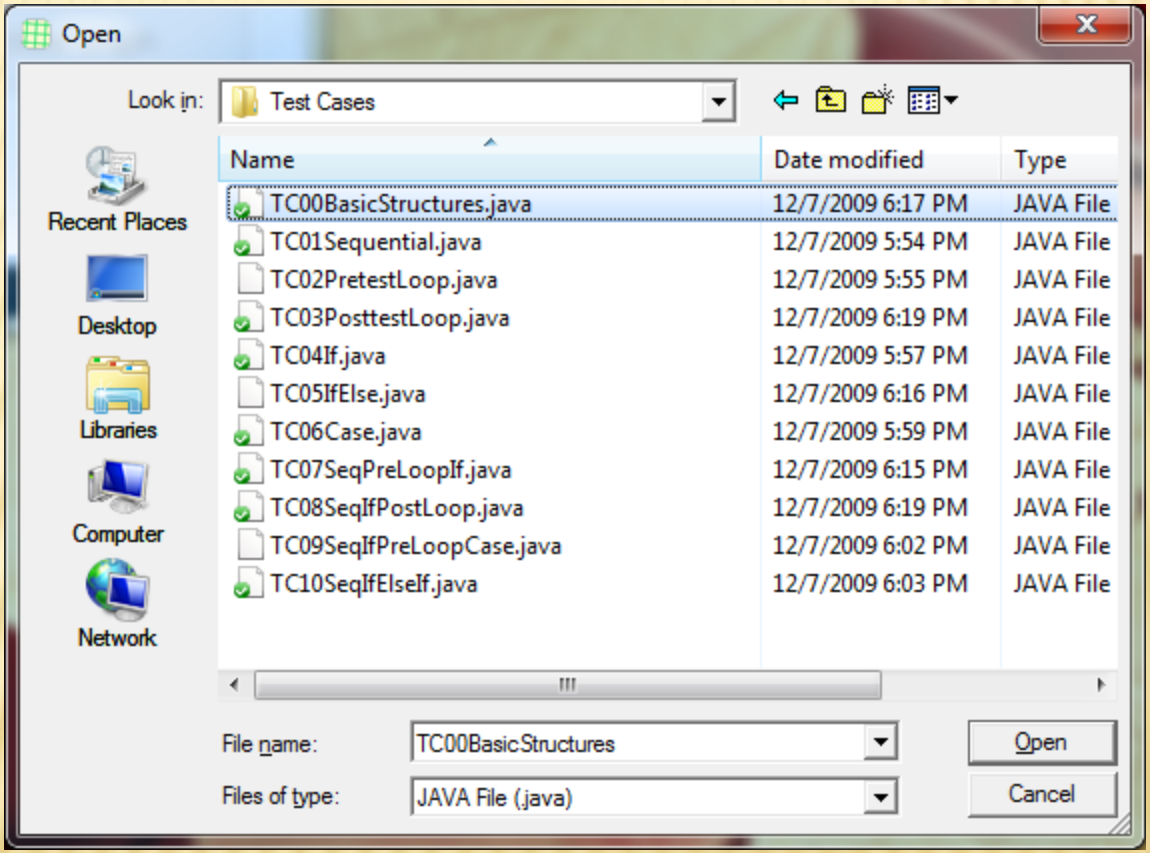
Password: cis310

1. Installation files
2. User manual
3. Sample test cases

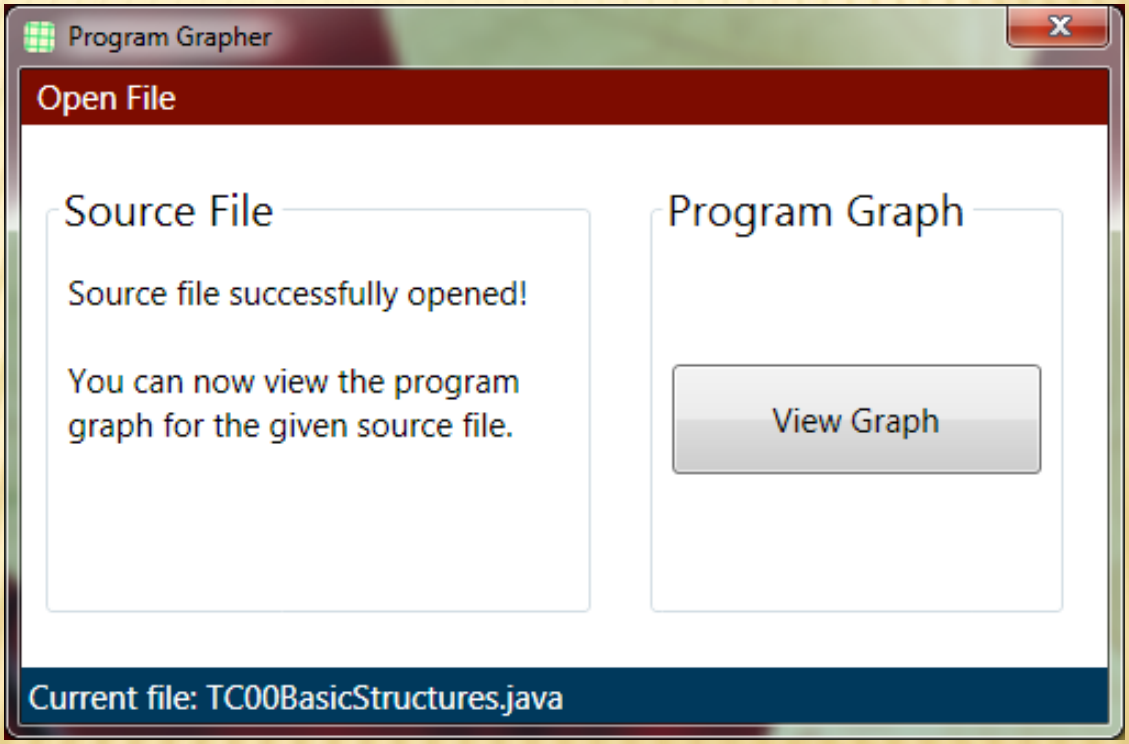
GUI Interface



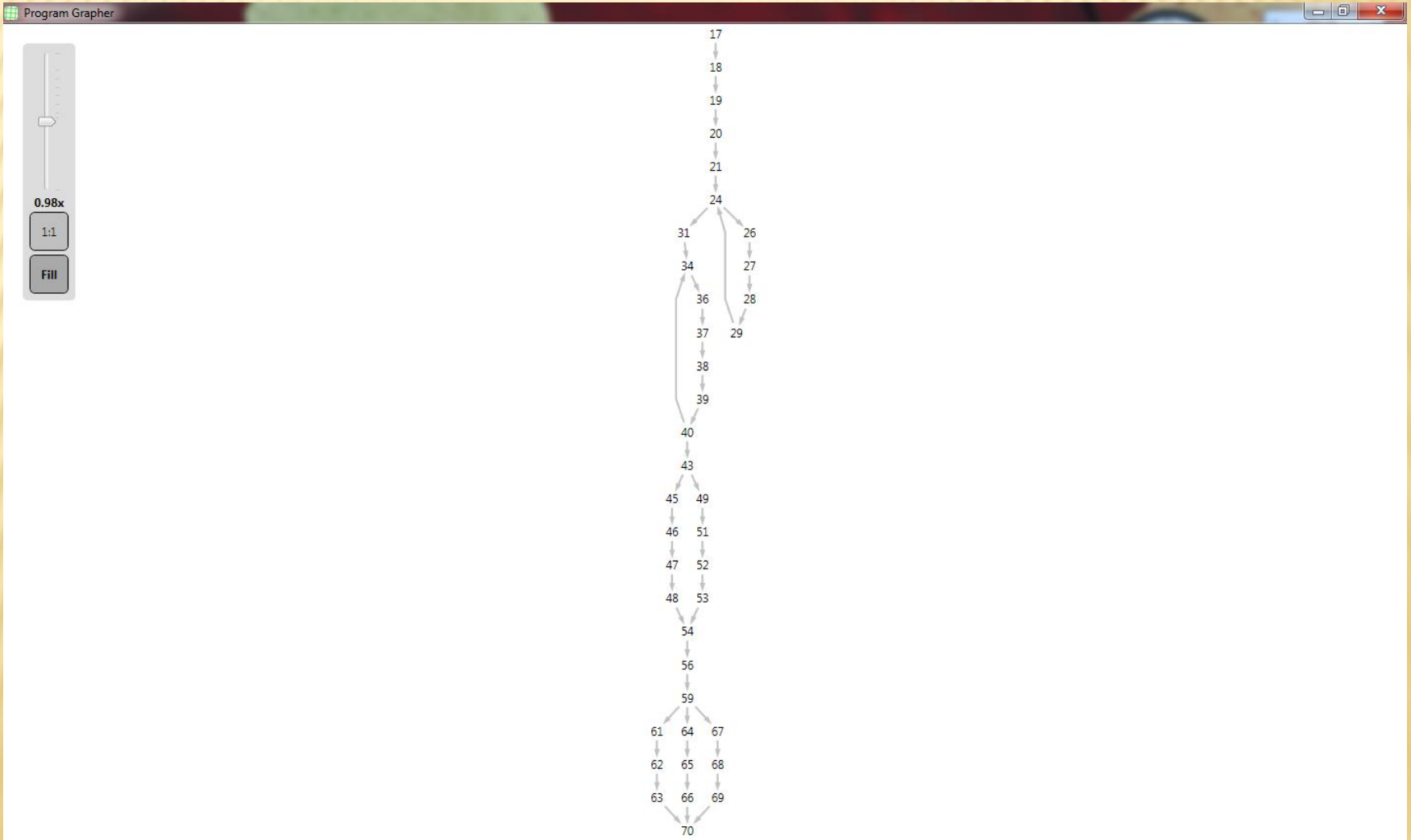
GUI Interface



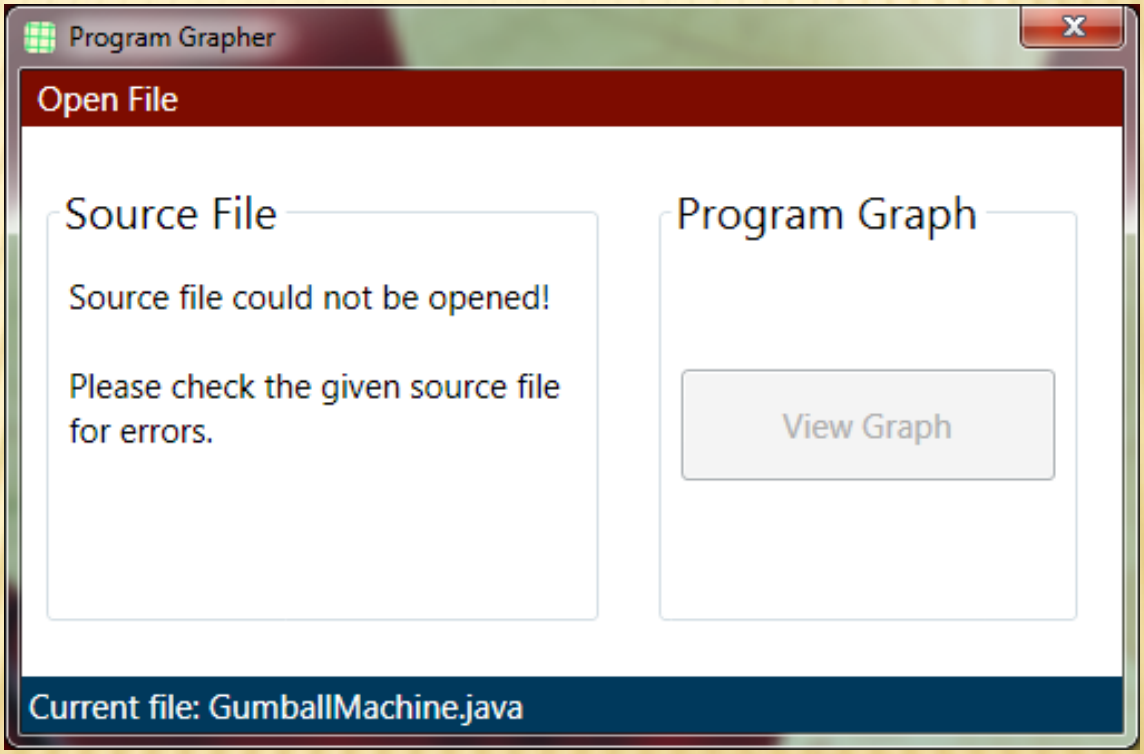
GUI Interface

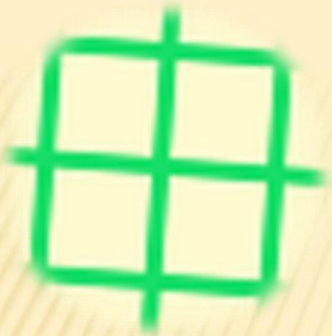


GUI Interface



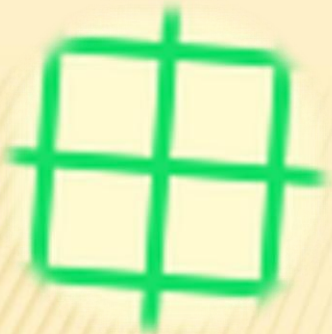
GUI Interface





Testing

- Functional Testing
  - 11 test cases covering expected behaviors and all necessary structures.
- Usability Testing
  - Study using 5 participants to operate the Program Grapher and report on their experience.



Functional Testing

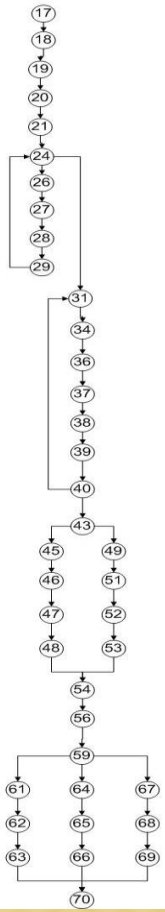
- 11 test cases were created by writing simple java source code files to test structures and functionality
- Expected results were created manually in Microsoft Visio 2010 from test case source code
- Test case source code was run through Program Grapher then compared with expected result graphs

Testing Example

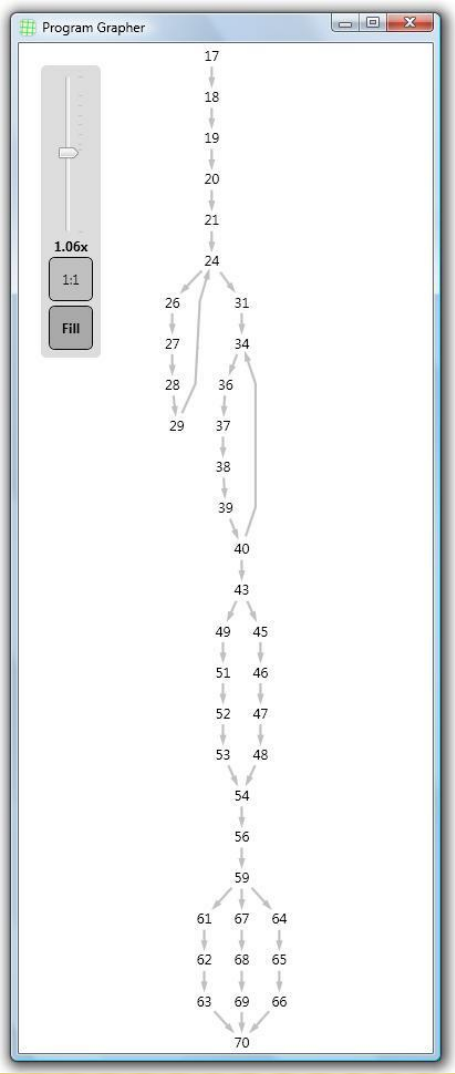
```java
// CIS 310/GCIS 567 Fall 2009
// Final Project
// Program Grapher Testing
// K. Drew Saxton
// Test Case #0: Basic Structures
// Test case includes all basic structures: sequential, pretest and posttest loops, if-else, case

package ProgramGrapherTests;
import java.io.*

public class driver {


    public static void main(String[] args) {

        // Sequence
        int intCounter = 0;
        intCounter++;
        intCounter++;
        intCounter++;
        System.out.println("The current count is" + intCounter);

        //Pretest Loop
        for(int x = 0; x < 10; x++)
        {
            intCounter++;
            intCounter++;
            intCounter++;
        }

        System.out.println("The current count is" + intCounter);

        // Post test Loop
        do
        {
            intCounter++;
            intCounter++;
            intCounter++;
            System.out.println("The current count is" + intCounter);
        } while (intCounter < 50);

        // If-Else
        if(intCounter < 5)
        {
            intCounter++;
            intCounter++;
            intCounter++;
        }
        else
        {
            intCounter--;
            intCounter--;
            intCounter--;
        }

        System.out.println("The current count is " + intCounter);

        // Switch-Case
        switch (intCounter)
        {
            case 1:
            System.out.println("The current count is one.");
            break;
            case 2:
            System.out.println("The current count is two.");
            break;
            default :
            System.out.println("The current count is neither one nore two.");
            break;
        }
    }
}
```

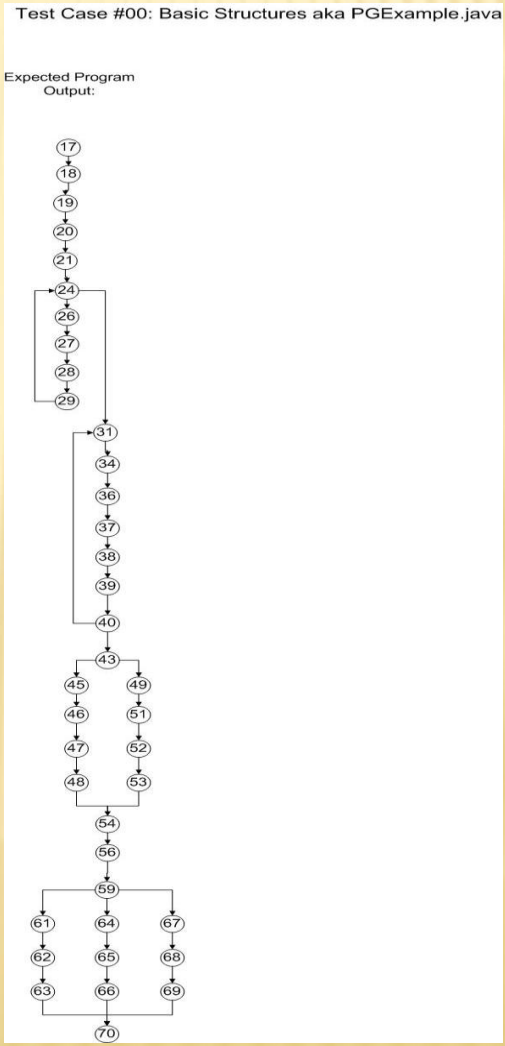
Test Case #00: Basic Structures aka PGExample.java

Expected Program Output:



Actual vs Expected



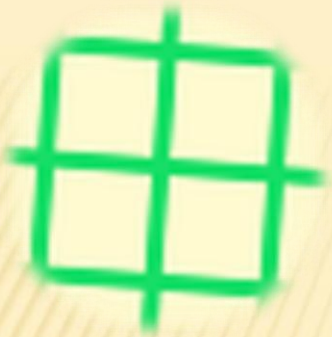


Functional Results

Case ID	Description	Passed?	Issues	Test Report File Name
TC00	Covers all structures	Yes	None	TC00Report.pdf
TC01	Sequential only	Yes	None	TC01Report.pdf
TC02	Pretest Loop	Yes	None	TC02Report.pdf
TC03	Posttest Loop	Yes	None	TC03Report.pdf
TC04	If	Yes	Displays odd line	TC04Report.pdf
TC05	If Else	Yes	None	TC05Report.pdf
TC06	Case	Yes	None	TC06Report.pdf
TC07	Sequential-Pretestloop-If	Yes	None	TC07Report.pdf
TC08	Sequential-If-Posttestloop	Yes	None	TC08Report.pdf
TC09	Sequential-If-Pretestloop-Case	Yes	None	TC09Report.pdf
TC10	If-Else-If-Else	Yes	None	TC10Report.pdf

Usability Testing

- Methodology: self report survey
- Materials: installation package, user manual
- Incentive: participants were promised cookies
- Participants: 5 professionals in computer science related fields
- Analysis: compiled and filtered using Microsoft Access 2010



Survey Caveats

- Personal relationship with developer may influence willingness to report negative findings.
- Lack of experience with program graphs in general may negatively impact responses.
- Self report surveys may be influenced by participants desire to appear knowledgeable
  - Less willing to report errors or failures

Usability Survey

Usability Testing Survey

Software Information:
Product Under Test: Program Grapher
Version: 1.0.0
Survey distributed: December 8, 2009
Researcher: K. Drew Saxton

System Information:
Date:
Computer Model:
Operating System: (Windows XP, Windows Vista, Windows 7)
RAM:
Video Card:

Participant Information:
Participant ID:
Participant Occupation:
Years in field:
Are you familiar with the Java programming language? Yes/No
Are you familiar with the topic of program graphing? Yes/No

Thank you for agreeing to participate in usability testing for the application "Program Grapher." Your honest evaluation will help us improve the current application and ensure it works as intended. Please be sure to review the minimum system requirements and other warnings included in the user manual before proceeding.

About the software: Program Grapher generates a program graph of Java files written in formal Java source code according to Sun Microsystems' guidelines.

Task One: Generate a Program Graph
Using the product manual, please generate a program graph from the example file "PGexample.java"

Were you able to complete this task using the manual alone? (Yes/No)

Please explain any errors encountered and attach screenshots if able:

Procedures to generate a program graph were: (select one)
Precise 7 6 5 4 3 2 1 Incomprehensible N/A

Button and controls are: (select one)
Intuitive 7 6 5 4 3 2 1 Incomprehensible N/A

Output generated by the program is: (select one)
Intuitive 7 6 5 4 3 2 1 Incomprehensible N/A

Overall the software is: (select one)
Intuitive 7 6 5 4 3 2 1 Incomprehensible N/A

Suggested Features and Functionality:

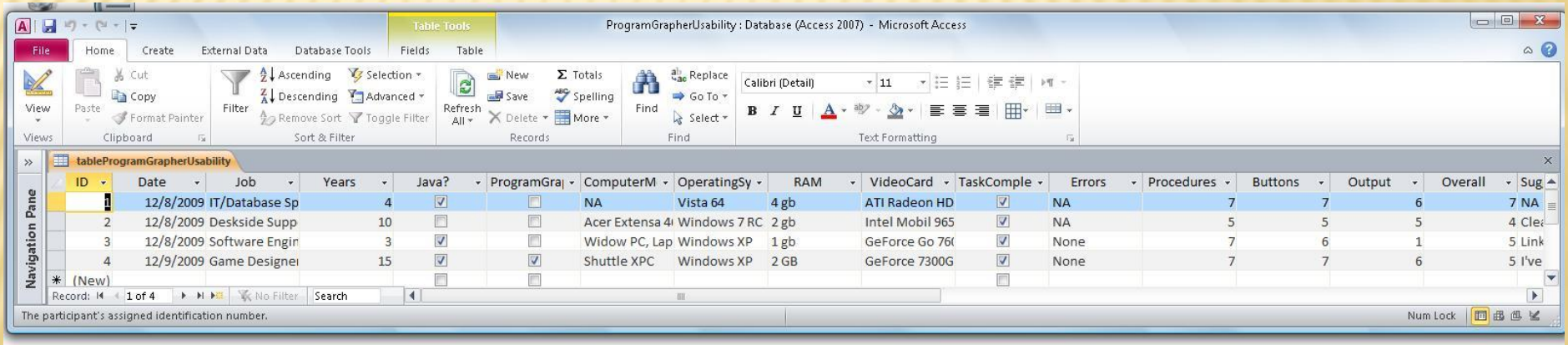
Additional Comments:

Thank you!

Usability Raw Data

Responses from the Usability Survey were collected in a Microsoft Access Database

- This allows rapid comparison and filtering of results by queries
- Can choose to view only results from most experienced participants
- Can use SQL to create issue, feature and comment reports
- Can use reports to analyze and detect subjective trends from Likert data



Survey Participants

All participants were selected from personal acquaintances of the developers with experience in Software Development or Software Support/Maintenance.

Most participants have experience with Java, but only one with program graphs.

ID#	Profession	Years	Java?	Graphs?
1	IT/Database Specialist	4	Yes	No
2	Deskside Support	10	No	No
3	Software Engineer	3	Yes	No
4	Game Designer	15	Yes	Yes
5	Research Assistant	3	No	No

Usability Results

All participants were able to open and generate a graph for the example using only the User Manual.

No participants experienced errors or failures.

A 1 to 7 (7 is best) Likert Scale records each participants' subjective experience with procedures, buttons and controls, output, and overall software quality.

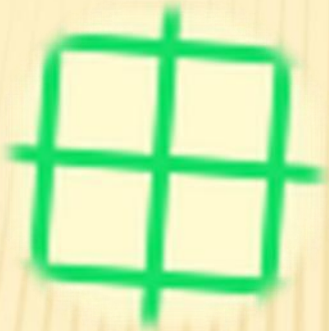
ID#	Complete?	Errors?	Procedures	Buttons	Output	Overall
1	Yes	No	7	7	6	7
2	Yes	No	5	5	5	4
3	Yes	No	7	6	1	5
4	Yes	No	7	7	6	5
5	Yes	No	7	6	7	5

Usability Issues

Issue	Response
I: Clearer instructions, add more screenshots	Added more steps with screenshots to manual
I: Open button does not look like button until mouse hovers over it	Change button style
I: Manual unclear about closing procedures	Updated wording for clarity across Windows variants

Feature Request	Response
FR: Links to information on program graphing	Added link to description and examples of Data Flow Testing and program graphing
FR: Ability to edit contents of each node	None
FR: Represent variables, definition and usage	None
FR: Ability to drag and drop source files to open	None

Program Grapher



Questions?